Integrating Performance Analysis in the Context of LOTOS-Based Design

INTEGRATING PERFORMANCE ANALYSIS IN THE CONTEXT OF LOTOS-BASED DESIGN

M. Ajmone Marsan† A. Bianco† L. Ciminiera* R. Sisto* A. Valenzano*

† Dipartimento di Elettronica
* CENS and Dipartimento di Automatica e Informatica
Politecnico di Torino Corso Duca degli Abruzzi, 24 10129 Torino ITALY

Abstract

Performance analysis and formal correctness verification of computer communication protocols and distributed systems have traditionally been considered as two separate fields. However, their integration can be achieved by using formal description techniques as paradigms for performance modeling. This paper investigates the possibility of using LOTOS, one of the two formal specification languages that have been standardized by ISO, as the formal basis for performance modeling. A LOTOS extension which encompasses both timing and probabilistic aspects is proposed, and a general performance model derivable from extended LOTOS specifications is identified. The performance model is open to different evaluation techniques. A simple example, a stop-and-wait protocol, is used to concretely demonstrate the new approach.

keywords: Specification languages, LOTOS, performance evaluation.

1 Introduction

Traditionally, formal correctness verification and performance analysis of communication protocols have been two separate fields. Whereas the validation and verification processes are based on formal techniques, the classical approach to performance analysis is based on human ingenuity and experience, and consists in devising specific abstract models which can be analyzed by simulation or by applying stochastic process theory. A key problem of the traditional performance evaluation approach lies in the credibility of the model: the functional equivalence of the performance model and the actual protocol is almost impossible to prove. This consideration suggests that formal description techniques should be used for performance analysis, besides formal verification. Such an integration brings along many other advantages. Among them, it makes performance prediction possible in the early design phases, thus avoiding costly redesign, and it facilitates the automation of the performance analysis process.

FDTs were originally conceived to describe the protocol behavior and functionalities in a time-independent fashion. However, the use of a formal description language as a paradigm for performance modeling requires the extension of the language with probabilistic and temporal specifications. The probabilistic specifications are necessary to describe the selection among different possible events, and the temporal specifications are necessary to describe the time lapse between consecutive events.

Many researchers have considered the two types of extension separately, the timing extension being interesting in itself for formal verification of time-critical systems, and the probabilistic extension being interesting for probabilistic verification when exhaustive verification is impossible. Their work gives a reasonable insight into the related problems, but merging probabilistic and timing information for performance modeling involves new aspects.

Pioneering work in this respect was done in the area of Petri Nets, with the formulation of some well known timed and probabilistic extensions such as Stochastic Petri Nets (SPN) [2, 3, 4] and their offsprings [5, 6, 7], or Timed Petri Nets [8, 9]. Other interesting proposals for integrating performance analysis within the framework of formal specification techniques have been made using state machine models or formal grammar models [10, 11].

In this paper we focus our attention on the FDT LOTOS [1], which is one of the two specification languages standardized by ISO. In this context, Rico and Von Bochmann proposed a method for introducing deterministic times and probabilities in LOTOS, so as to obtain a corresponding semi-Markov model which can be used for performance analysis [12]. Their timing model however is not able to describe global timing constraints properly.

In this paper we describe a more general extension of LOTOS that is suitable for the construction of performance models of communication protocols, and distributed systems. The proposed version of extended LOTOS is derived from a previous timed extension proposal by Bolognesi et al. [13], but it considers some new features, such as the introduction of random variables for the probabilistic specification of timing, and the use of priorities and weights for the resolution of conflicts.

The reader is assumed to be familiar with LOTOS. A comprehensive tutorial can be found in [14].

0-8186-5292-6/94 $3.00

2 The LOTOS Extension

T-LOTOS [13] is a timed LOTOS extension which mainly provides LOTOS with a new operator called `timer`. In order to define the semantics of such operator, the age of an action prefix is defined as the cumulative amount of time during which the action prefix event has been continuously enabled. The expression

```
timer g<t1,t2> in B
```

indicates that in expression `B` the time interval `[t1,t2]` is assigned to gate `g`. The process described by `B` may execute an action at `g` only when the age of such action lies in the time interval `[t1,t2]`. Note that `g` may be a gate at which several subprocesses of `B` synchronize, in which case the temporal specification assumes a global significance over all the subprocesses in `B` (aging starts when the last subprocess gets ready at `g`.

Our extension, that we call Extended Timed LOTOS (ET-LOTOS), is based on T-LOTOS, but we admit the use of timer operators (including the extensions that will be introduced) only with a global scope, and we forbid the use of cascaded timer operators. The effect of such requirements is that each timed event is controlled by a single timer operator, and the assignment of timings is clear and easy to understand.

Let us now describe the extensions of our proposal with respect to T-LOTOS.

2.1 Pre-synchronization timer

The definition of T-LOTOS enables the specifier to directly express situations in which the processes interacting at a gate, after reaching a consensus on the synchronization, must wait for a given time before actually being committed to the synchronization. During this time interval, it is possible for each one of the processes involved in the synchronization to execute another action, thus aborting the synchronization.

In some cases, instead, it may be useful to be able to express situations in which the commitment is instantaneous, but a certain time is necessary to perform common operations. Only after this time has elapsed, the involved processes are free to execute other actions. This different kind of timed synchronization can simply be expressed in terms of the basic synchronization introduced in [13]. In order to do this, it is necessary to split the synchronization into a pre-synchronization event, which is an interaction with null delay, followed by the timed synchronization. For example, if we want to give this new meaning to the synchronization expressed by the following expression

```
timer g<2,3> in (a; g; stop |[g]| g; b; stop)
```

we can write

```
hide h in timer h<0,0>, g<2,3> in
(a; h; g; stop |[h,g]| h; g; b; stop)
```

where

```
timer X,Y in B = timer X in timer Y in B
```

Since the modifications necessary to implement this new kind of synchronization may be extensive, and may complicate the specification significantly, resulting in reduced readability, it seems useful to introduce a new timer operator, whose semantics can be expressed in terms of the original timer operator semantics. We chose to define the new operator as a macro expansion operator, which will be indicated by the new keyword `p_timer`, and will be assigned the same syntax as the `timer` operator. The macro expansion rule therefore is:

```
p_timer g<t1,t2> in B =
hide g' in timer g'<0,0>, g<t1,t2> in B'
```

where `g'` is a new unique identifier and `B'` is the same as `B` with all occurrencies of action prefix `g;` replaced by `g';` `g;`, and all occurrencies of `g` in synchronization operators replaced by `g',g`, the same replacement being recursively applied to all the processes directly or indirectly instantiated in `B`.

2.2 Probabilistic extension

In T-LOTOS, the temporal properties of a system are described using the `timer` construct which assigns to a gate `g` a time interval within which any action on `g` must be executed.

We introduce random variables to describe the time lapse according to the same temporal specification, but in a probabilistic way. For each random variable it is necessary to define a time interval, which is the domain of the random variable, and a probability density function of the random variable. The first item is the time interval associated with the corresponding gate as in T-LOTOS, while the second item is an additional information which should be assigned to each time interval. Whenever an action becomes enabled, a random variable instance is extracted from the probability density function, and its value represents the actual delay of the action; the instance value lies in the domain of the random variable, as required by T-LOTOS semantics.

This temporal specification describes time lapse in an explicit way, and probabilistic selection in an implicit way, because, according to T-LOTOS semantics, the action whose associated delay is shortest, is actually selected for execution when several actions are competing. Nevertheless, the explicit definition of a selection criterion is sometimes mandatory, e.g., to solve a conflict between enabled events with the same deterministic delay, or a conflict between events with associated delays described as random variables when the instances of the random variables are identical. We introduce a priority (ranging from p_{min} to p_{max}) and a weight (ranging from w_{min} to w_{max}) associated with each event. When a choice between enabled events with the same delay must be made, the priority and the weight are orderly used to solve the conflict: first of all, the priorities are examined, and a deterministic decision is taken; if the conflict is not solved (i.e., two or more enabled events have the same priority level), the enabled events are assigned probabilities proportional to their weight, and a random choice is made according to this probability assignment.

Priorities and weights can be assigned to gates just as time intervals are assigned to gates in [13]. The resulting syntactic extension to the `timer` operator is the following:

```
timer  a<t1,t2,pdf(),priority,weight>  in B
```

where a is a gate, t1,t2 (t2 $\geq$ t1) describe the domain of the random variable representing the delay associated with the gate, pdf() is a function $[t1, t2] \rightarrow [0, \infty]$ that describes the characteristic of the pdf ($\int_{t1}^{t2} \mathrm{pdf}(x)dx = 1$) of the same random variable, priority is the priority used to solve conflicts in a deterministic way, weight is the weight used to solve in a probabilistic way the conflicts not solved by priorities, and B is a behavior expression.

A default time interval <0,0> (deterministic null time), a default priority $\frac{p_{max} + p_{min}}{2}$ and weight $\frac{w_{max} + w_{min}}{2}$ are assigned to a gate when they are not explicitly defined.

The extensions with respect to the T-LOTOS proposal do not alter the possible behaviors of the specification, but only make it possible to specify that certain behaviors are more likely to occur than others. The semantics expressed in [13] are therefore still valid for the purpose of verification and, in general, when the probabilistic aspect is not of interest.

2.3 Memory timer

When modeling communication protocols and distributed systems, it is important to be able to describe different typical scenarios.

In general, a system in a given state can be considered as a set of parallel activities requiring some (residual) time to complete. In T-LOTOS, each activity is a gate interaction, and a timer operator is used to express the time it takes from the instant it is enabled until the instant it completes. The age associated with actions and interactions represents the amount of time the activity has already spent, i.e., the amount of work already performed by the activity.

The simplest scenario is a memoryless one, in which the occurrence of any event has the effect of restarting all the parallel activities in the system. In practice, if more than one event is enabled, the one which takes place first causes the work performed by the other activities up to that instant to be lost. This scenario corresponds to rather uncommon situations, since normally in distributed systems each process keeps its memory when events occur in other concurrent processes.

The `timer` operator introduced in [13] enables the user to model a more realistic scenario, i.e., one in which the occurrence of an event does not restart all the activities: concurrent activities not synchronized with the one(s) having produced the event remain enabled after the state change and are not restarted, which means that the work they have already performed (i.e. their age) is not lost. Only the work performed by the other activities is lost: they will be restarted when they become enabled again.

Even if this kind of behavior is useful in modeling a great variety of real systems, there is however another typical scenario in distributed systems, which is not included in the timed model proposed in [13], but it is equally important. This scenario is one in which the occurrence of an event does not restart any activity, except the one associated with the event which has just taken place. Moreover, even if an activity which was enabled is no longer enabled in the new system state, the work it has previously performed is not lost: when the activity is enabled again later, it will resume its work from the point at which it was interrupted.

In order to model this different kind of situations within the framework of timed LOTOS, it is necessary for the timed model proposed in [13] to be enhanced, and this can be done by introducing a new `timer` operator, whose semantics is defined according to the new timed behavior. We shall call this new operator **a memory timer operator**, and we shall assign it a syntax similar to the one used for the `timer` operator in [13], incorporating the extensions introduced in the previous section:

```
m_timer a<t1,t2,pdf(),priority,weight>  in B
```

The introduction of this new operator modifies the semantics of the LOTOS extension that we have considered up to now. The new semantics are not reported here for lack of space.

2.4 Time-independent probabilistic choice

Expressing the probabilistic characterization of a nondeterministic choice by assigning specific time distributions to the various alternatives is satisfactory in some situations, but it may lead to difficulties in other contexts, such as when the timing information is naturally independent of the selection criterion. For example, suppose that we must model a communication channel with a certain failure rate and with a certain distribution of transmission time. A proper time distribution must be assigned to the message loss event in such a way that the failure rate takes the desired value, but this is not straightforward, and forces the specifier to introduce a fictitious delay in the message loss event.

The introduction of pre-synchronization (the `p_timer` operator) helps in solving this difficulty: if the initial events representing the various alternatives are subject to `p_timer` operators (or if they are characterized by identical deterministic times), their priorities and weights determine a choice which is independent of the delay assignments.

3 A General Framework for Performance Modeling

In this section we describe a general environment for the construction of performance evaluation models onto which ET-LOTOS specifications can be easily mapped.

The model is based on a state machine, and is composed of the following entities:

- a set $S = \{s_i \mid 1 \leq i \leq M\}$ of states.

- a set $T = \{t_j \mid 1 \leq j \leq N\}$ of transitions, divided into two classes: transitions without memory ($U \subseteq T$) and transitions with memory ($V \subseteq T$);
- a mapping $E : S \rightarrow \mathcal{P}(T)$ ($\mathcal{P}$ denoting the power set operation), which defines, for each state s_i, the set of transitions enabled in that state, $E(s_i)$;
- a new-state partial function $N : S \times T \rightarrow S$, such that $N(s_i, t_j)$ is the state reached by firing transition t_j when in state s_i. The function is defined only for (s_i, t_j) pairs such that $t_j \in E(s_i)$;
- a mapping $Q : S \times T \rightarrow \mathcal{P}(T)$ which defines, for each pair s_i, t_k such that $t_k \in E(s_i)$, the set $Q(s_i, t_k)$ of transitions whose associated delay must be resampled as a consequence of firing t_k in state s_i;
- a set D of random variables, $D = \{d_j \mid 1 \leq j \leq N\}$, d_j representing the delay associated with transition t_j;
- a set R of variables, $R = \{r_{ij} \mid 1 \leq i \leq M, 1 \leq j \leq N\}$, r_{ij} representing the residual delay associated with transition t_j in state s_i;
- a set P of priorities $P = \{p_j \mid 1 \leq j \leq N\}$, p_j being the priority assigned to transition t_j;
- a set W of weights $W = \{w_j \mid 1 \leq j \leq N\}$, w_j being the weight assigned to transition t_j.

The evolution of the system consists in state changes and in the elapsing of time. Each state change corresponds to a transition and is instantaneous.

When the system is in state s_i, each transition t_j has an associated residual delay r_{ij}. If there exists a transition t_k enabled in state s_i (i.e., $t_k \in E(s_i)$) whose residual delay in state i, r_{ik}, is such that

$$r_{ik} < r_{ij} \quad \forall j \mid t_j \in E(s_i),\ j \neq k \tag{1}$$

then t_k occurs when its associated delay r_{ik} has elapsed and its occurrence determines the next system state $l = N(s_i, t_j)$.

If two or more residual delays in the set of enabled transitions assume the same minimum value, priorities and weights are used to establish which transition occurs, exactly as priorities and weights are used in ET-LOTOS to determine the next event when two or more events are ready to occur. More precisely, first priorities are used, and transitions t_k such that

$$r_{ij} > 0\ \forall j \mid t_j \in E(s_i),\ p_j > p_k,\ j \neq k \tag{2}$$

are selected. Finally, if more transitions satisfy equation (2), transition t_k is chosen with a probability

$$p = \frac{w_k}{\sum_{x \in X_i} w_x} \tag{3}$$

with $X_i = \{x \mid t_x \in E(s_i)\}$.

The residual delays r_{lj} of the new set of enabled transitions in state l are computed as follows:

- for all transitions t_j without memory ($t_j \in U$)
  - $r_{ij} - r_{ik}$, if t_j ($j \neq k$) was enabled in state s_i, is still enabled in state s_l, and $t_j \notin Q(s_i, t_k)$, (i.e. its delay must not be restarted when t_k fires;
  - d_j^*, if t_j was disabled in state s_i and is enabled in state s_l, or if $j = k$ and transition t_k is enabled in state s_l, or if the transition was enabled in state s_i, is still enabled in state s_l, and $t_j \in Q(s_i, t_k)$ (d_j^* represents a new instance of the random variable d_j).
  - $-\infty$, if t_j is not enabled in state s_l.
- for all transitions t_j with memory ($t_j \in V$)
  - $r_{ij} - r_{ik}$, if t_j ($j \neq k$) was enabled in state s_i;
  - if t_j was disabled in state s_i it keeps its associated residual delay value.
  - d_k^*, if $t_j = t_k$ (d_k^* represents a new instance of the random variable d_k).

In order to map an ET-LOTOS specification onto a performance model of the kind just described, a finite state machine model equivalent to the specification must be built. Of course this is not always possible, as a LOTOS specification can be characterized by an infinite number of states, but, given such a state machine, the mapping is straightforward, as ET-LOTOS events correspond to the performance model transitions, and, since a timer operator (possibly the default one) is assigned to each event, all the other timing and probabistic parameters are consequently defined. An algorithm for generating a state machine from a LOTOS specification, along with sufficient conditions assuring that the state machine is finite can be found for example in [15].

4 Performance model evaluation

The description of the dynamic behavior of the performance model into which a timed LOTOS specification can be mapped provides sufficient details for the implementation of a simulator that can be instrumental for the computation of the performance indices of interest. Of course, the extensions to the original timer constructs are such that the performance analysis can be obtained independently of any restriction on the probabilistic characterization of the temporal specification. It may however be interesting to discuss under what conditions an analytical approach to performance evaluation is possible and convenient.

First of all, it is important to note that by an analytical approach we mean the study of the stochastic model generated from the timed LOTOS specification by numerical methods, since the complexity of even the simplest toy examples immediately rules out the possibility of any closed-form solution.

Obviously, the use of exponential distributions for the characterization of the random variables associated with the action timers allows the mapping of the

timed LOTOS specification onto a continuous-time Markov chain. This opens the possibility of using the numerical tools developed in many years of lively research in the field of efficient numerical techniques for the steady-state solution of Markovian models. The present state of the art in this field is such that models comprising few hundred thousand states can be analyzed with acceptable time and space complexity.

The adoption of probability distributions formed by adequately combining exponential stages also leads to Markovian models, but in this case the number of states of the Markov chain is much larger than the number of states of the timed LOTOS specification. Similarly, by introducing some (tight) restrictions on the use of general distributions in combination with a majority of exponentially distributed timers would lead to semi-Markov models, that in principle can be analyzed, but at very high cost.

In summary, this means that a numerical approach to the analysis of the performance model derived from a timed LOTOS specification seems feasible only in the case of exponential timing and "reasonable" state space sizes. In all other instances, simulation is probably more convenient. This statement should not be interpreted as a claim that the simulation of extraordinarily large models is simple, regardless of the timing specifications; on the contrary, the model size makes obtaining reliable performance estimates extremely difficult, but no simple alternative is known.

5 Example: a stop and wait protocol

In this section we present a model of the stop-and-wait protocol (the example that is always used in the literature), which provides an application example of the performance evaluation approach based on a continuous-time Markov chain and makes a model validation possible (by comparison with results found in the literature).

We consider a stop-and-wait protocol with one bit frame numbering, and the timeout mechanism at the transmitter. The ET-LOTOS specification of the protocol is:

```
Specification stopwait[tf0,tf1,ra0,ra1,rf0ta0,rf1ta1,
timeout]:noexit

Behaviour

timer   tf0 <0,infty,exp(13.47)> in
timer   tf1 <0,infty,exp(13.47)> in
timer   timeout <0,infty, exp(1000)> in
timer   rf0ta0 <0,infty, exp(120.14)> in
timer   rf1ta1 <0,infty, exp(120.14)> in
p_timer ra0 <0,infty, exp(106.7)> in
timer   ra1 <0,infty, exp(106.7)> in

TX[tf0,tf1,ra0,ra1,timeout]
|[tf0,tf1,ra0,ra1,timeout]|
(TIMER[tf0,tf1,timeout]
  |[tf0,tf1]| CH[tf0,tf1,rf0ta0,rf1ta1,ra0,ra1])

where

process TX[tf0,tf1,ra0,ra1,timeout]:noexit :=
tf0; WA[tf0,tf1,ra0,ra1,timeout]
endproc

process WA[tf0,tf1,ra0,ra1,timeout]:noexit :=
   timeout; TX[tf0,tf1,ra0,ra1,timeout]
[] ra0; TX[tf1,tf0,ra1,ra0,timeout]
[]  ra1; WA[tf0,tf1,ra0,ra1,timeout]
endproc

process TIMER[start0,start1,timeout]:noexit :=
   start0; SET_TIMER[start0,start1,timeout]
[] start1; SET_TIMER[start0,start1,timeout]
endproc

process SET_TIMER[start0,start1,timeout]:noexit :=
   start0; SET_TIMER[start0,start1,timeout]
[] start1; SET_TIMER[start0,start1,timeout]
[] timeout; TIMER[start0,start1,timeout]
endproc

process CH[tf0,tf1,rf0ta0,rf1ta1,ra0,ra1]:noexit :=
CH1[tf0,tf1,rf0ta0,rf1ta1]
 |[rf0ta0,rf1ta1]| CH1[rf0ta0,rf1ta1,ra0,ra1]
endproc

process CH1[t0,t1,r0,r1]:noexit :=
hide ok, err in
timer ok  <0, 0,,, 95>, err <0, 0,,, 5> in
   t0; (ok; r0; CH1[t0,t1,r0,r1]
        [] err; CH1[t0,t1,r0,r1])
[] t1; (ok; r1; CH1[t0,t1,r0,r1]
        [] err; CH1[t0,t1,r0,r1])
endproc

endspec
```

where exp(x) indicates an exponential pdf with mean value x.

The protocol behavior is modeled by three parallel processes: a transmitter TX, a communication channel CH, which acts both as channel and receiver, and a TIMER, which models the timeout mechanism.

Gates tf0 and tf1 represent the transmission of a frame with sequence number 0 and 1 respectively; gates rf0ta0 and rf1ta1 represent the propagation and the reception of a frame, and the corresponding acknowledgment transmission at the receiver; gates ra0 and ra1 represent the acknowledgment propagation and reception; gate timeout represents the timeout expiration at the transmitter.

After the frame transmission, TX instantiates the WA (Wait for Acknowledgment) process, which discards the ACKs referring to out of order frames (i.e. interactions offered at gate ra0 while waiting for an acknowledgment at gate ra1 or vice versa), until a correct ACK is received or the timeout occurs. If a correct ACK is received, process TX is restarted, but now gate tf1 takes the place of gate tf0 (and vice versa), and the same holds for gates ra0 and ra1, so as to obtain the correct frame numbering. If a timeout occurs (i.e. an interaction at gate timeout takes place) before the correct acknowledgment is received, process TX is restarted without inverting the gates, since the same frame will be transmitted again.

Process TIMER models a timer which is set whenever

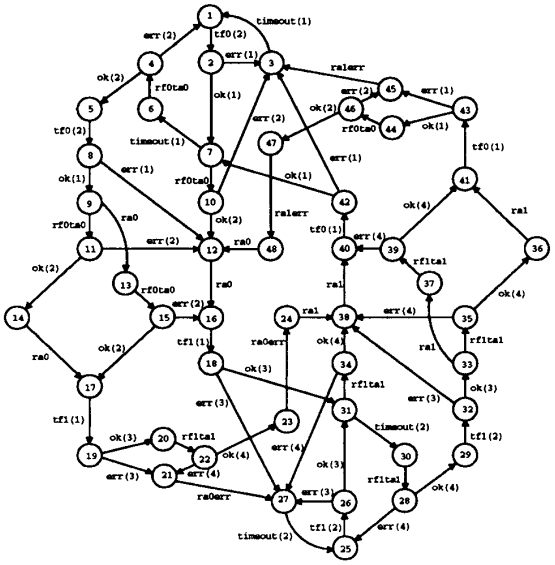

Figure 1: The performance model state machine for the stop-and-wait protocol example

a frame is transmitted (an interaction at gate `tf0` or gate `tf1` takes place), and expires after an exponentially distributed time, if not reset by a new frame transmission. The choice of an exponential distribution comes only from the desire to make a Markovian analysis possible. Should performance analysis be done by simulation, more realistic distributions could be used.

Process `CH` is composed of two parallel processes, each one representing a unidirectional channel: one conveys frames, while the other conveys acknowledgments. After a frame or ACK transmission (`t0` or `t1`), the channel can lose a frame with probability 0.05 (event `err`) or successfully propagate it (event `ok`) and offer it at the corresponding gate (`r0` or `r1`). The use of our extension allows a direct specification of the 5% channel error probability, simply using appropriate weights for gates `err` and `ok`. If an error occurs on the channel, the `timeout` event takes place at the transmitter because no other interaction can occur.

Note the use of a pre-synchronized timer on gates `ra0` and `ra1` to prevent the fact that action `timeout` can take place before the incorrect pending acknowledgment is discarded. This behavior must be excluded in order to avoid a possible deadlock.

Note that, due to the exponential pdf on gates `rfa0ta0` and `ra0`, a timeout can occur also if the frame and ACK transmissions take place without channel error but the process is so slow that the timeout expires. This is a realistic scenario that we can describe using ET-LOTOS. Moreover, the use of global timers greatly improves the readability of the specification.

The performance model is composed of 48 states and 68 transitions. The corresponding state machine

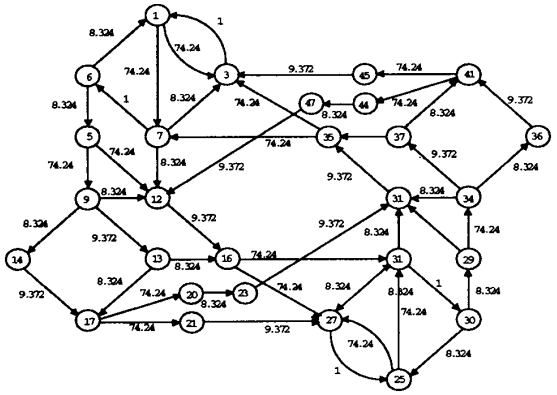

Figure 2: The continuous-time Markov chain transition rate diagram for the stop-and-wait protocol

is shown in Fig. 1. Transitions are labeled with the corresponding gates, while states are numbered starting from 1, which indicates the inital state. As each possible event is mapped onto a performance model transition, and several conceptually different events can correspond to a single gate, integer indexes have been added in order to distinguish among different instances of an event at a given gate (for instance, `tf0(1)` and `tf0(2)`).

Since all the probability density functions of the random times are exponential a markovian analysis is possible, based on a continuous-time Markov chain whose state transition rate diagram is shown in Fig. 2. It can be observed that the graph is similar to the one given in Fig. 1: the same state numbers used for the state machine are also used here, but zero-delay transitions and vanishing states do not appear in the transition rate diagram. Transitions are labelled with transition rates, whose values are expressed in s^{-1}.

In [4], a model of the stop-and-wait protocol with exponentially distributed times is analyzed. In order to compare the results obtained with our approach with the ones reported in [4], the same numerical values for the protocol parameters must be used. However, as the two models are slightly different, our model parameters do not map one to one onto corresponding parameters used in [4], and this difference must be taken into account. In [4] one transition is used to describe the frame transmission and propagation times, and a separate transition is used for the frame reception time (the same is true for the ACK); in our model, instead, we describe as three separate actions the frame transmission, the frame propagation, reception and ACK transmission, and the ACK propagation and reception.

A proper interpretation of the values used in [4] gives for our model the following corresponding values, used in the ET-LOTOS specification: a frame transmission time (gates `tf0` and `tf1`) with mean value equal to 13.47 ms; a frame propagation, reception, and ACK transmission time (gates `rf0ta0` and

rf1ta1) with mean value 120.14 ms; an ACK propagation and reception time (gates ra0 and ra1) with mean 106.7 ms; a timeout with mean 1000 ms. The error probability on the channel is 0.05 in both models.

The Markovian analysis with numerical techniques gives a protocol throughput equal to 2.7532 messages/s, quite close to the value 2.75 reported in [4]. The difference in the numerical values is due to the different ways in which the error probability on the channel is described: in [4], an exponential distribution is used to induce the error probability, while we describe it explicitly via a probabilistic choice. Moreover, in [4] a timeout is set only if an error occurs on the channel, while in our model we describe a more realistic timeout mechanism.

6 Conclusions

"Extended timed LOTOS" (ET-LOTOS) is a new extension of LOTOS which incorporates both timing and probabilistic specifications. Extended timed LOTOS maintains the same formal structure of LOTOS, and can therefore be similarly used to formally verify distributed systems, including those with time-critical features. The extension is downward compatible, in the sense that by neglecting extensions one gets a standard LOTOS specification describing the system features not related to time or probability.

ET-LOTOS can be mapped onto a performance model which is open to direct application of different performance evaluation techniques. The choice about which specific technique it is possible and convenient to use depends on the kind of timing and probabilistic characterizations used in the specification.

The extensions introduced in the language enable the user to properly and easily specify the most common scenarios in distributed systems, including some which were not provided for in previous timed extensions of LOTOS.

References

[1] IS 8807: *Information Processing Systems, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, ISO, 1989.

[2] F.J.W. Symons, "Introduction to Numerical Petri Nets, a General Graphical Model of Concurrent Processing Systems", *Australian Telecommunications Research*, Vol. 14, n. 1, pp. 28-33, January 1980.

[3] G. Florin, and S. Natkin, "Les Reseaux de Petri Stochastiques", *Technique et Science Informatiques*, Vol. 4, n. 1, February 1985.

[4] M. K. Molloy, "Performance Analysis using Stochastic Petri Nets", *IEEE Trans. on Computers*, Vol. 31, n. 9, pp. 913-917, September 1982.

[5] M. Ajmone Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems", *ACM Transactions on Computer Systems*, Vol. 2, n. 1, May 1984.

[6] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis", *Proc. PERFORMANCE '84*, Paris, France, December 1984.

[7] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application", *Proc. Int. Workshop on Timed Petri Nets*, IEEE-CS Press, Torino, Italy, July, 1985.

[8] R.R. Razouk, and C.V. Phelps, "Performance Analysis using Timed Petri Nets", *Proc. Int. Conf. on Parallel Processing*, pp. 126-129, August 1984.

[9] M.A. Holliday, and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis", *Proc. Int. Workshop on Timed Petri Nets*, IEEE-CS Press, Torino, Italy, July, 1985.

[10] H. Rudin, "An Improved Algorithm for Estimating Protocol Performance" in Y. Yemini, R. Storm, and S. Yemini (Eds.) *Protocol Specification, Testing and Verification IV*, Elsevier Science, 1985.

[11] F. J. Lin, and M. T. Liu, "An Integrated Approach to Verification and Performance Analysis of Communication Protocols", in S. Aggarwal and K. Sabnani (Eds.) *Protocol Specification, Testing and Verification VIII*, Elsevier Science, 1988.

[12] N. Rico, and G. Von Bochmann, "Performance Description and Analysis for Distributed Systems Using a Variant of LOTOS", in B. Jonsson, J. Parrow and B. Pehrson (Eds.), *Protocol Specification, Testing and Verification XI*, Elsevier Science, 1991.

[13] T. Bolognesi, and F. Lucidi, "LOTOS-like process algebras with urgent or timed interactions", in K. Parker and G. Rose (Eds.) *Proc. of FORTE'91: 4th Int. Conf. on Formal Description Techniques*, Elsevier Science, 1992.

[14] T. Bolognesi, and E. Brinksma, "Introduction to the ISO Specification Language LOTOS", *Computer Networks and ISDN Systems*, Vol.14 (1987), pp. 25-59.

[15] A. Valenzano, R. Sisto and L. Ciminiera, "An Abstract Execution Model for Basic LOTOS", *IEE Software Engineering Journal*, Vol. 5 (1990) No. 6, pp. 311-318.